

Sortialgorithmen im Vergleich

Eine Suche kann in einer sortierten Reihung nach dem Prinzip der binären Suche effizienter durchgeführt werden als eine Suche in einer unsortierten Reihung (vgl. AB01). Große Datenbestände, in denen häufig nach Werten gesucht werden muss, werden daher in der Regel sortiert. Da Datenbestände wie z. B. Kundennummern eines Versandhandels, Ferienhäuser einer Ferienhausagentur oder das Vorlesungsverzeichnis einer Universität, sehr umfangreich sein können, ist es von Interesse, einen möglichst effizienten Sortieralgorithmus zu verwenden.

In Aufgabe 1 bis 4 werden zunächst drei verschiedene iterative Sortieralgorithmen und ein rekursiver hinsichtlich ihrer Komplexität untersucht. Es bietet sich an, die Aufgaben arbeitsteilig zu bearbeiten (insbesondere die Aufgaben 1 bis 3) und anschließend die Ergebnisse zu sammeln (→ Aufgabe 5).

Aufgabe 1: Abbildung 1 zeigt den Sortieralgorithmus *SelectionSort* in Form eines Struktogramms.

- Veranschaulichen Sie die Arbeitsweise des Algorithmus am Beispiel der Reihung
daten: 15, 13, 14, 18, 17, 11
- Begründen Sie, dass die Laufzeit der inneren Schleife des Algorithmus linear abhängig von der Länge n der Reihung *daten* ist.
- Bestimmen Sie die Komplexitätsklasse für die Laufzeit des Algorithmus.

selectionSort(daten: Reihung vom Inhaltstyp Ganzzahl): Reihung vom Inhaltstyp Ganzzahl



Abbildung 1: Struktogramm des Sortieralgorithmus *SelectionSort*

Aufgabe 2: Abbildung 2 zeigt den Sortieralgorithmus *BubbleSort* in Form eines Struktogramms.

- Veranschaulichen Sie die Arbeitsweise des Algorithmus am Beispiel der Reihung
daten: 15, 13, 14, 18, 17, 11
- Begründen Sie, dass für eine Reihung der Länge n die äußere Schleife des Algorithmus höchstens n -mal ausgeführt werden muss.
- Bestimmen Sie die Komplexitätsklasse für die Laufzeit des Algorithmus im besten, im durchschnittlichen und im schlechtesten Fall.

bubbleSort(daten: Reihung vom Inhaltstyp Ganzzahl): Reihung vom Inhaltstyp Ganzzahl

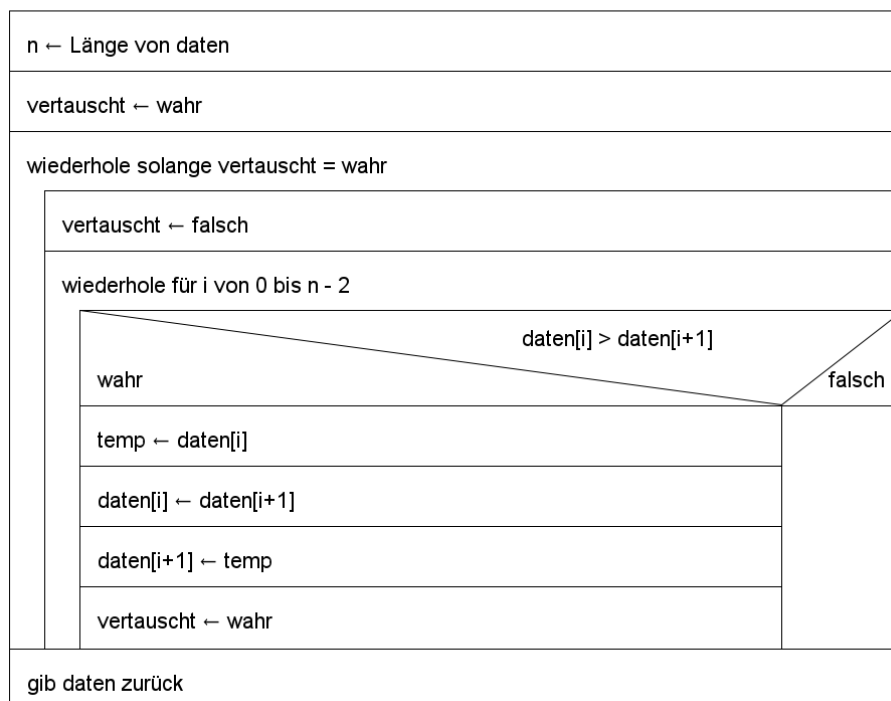


Abbildung 2: Struktogramm des Sortieralgorithmus *BubbleSort*

Aufgabe 3: Abbildung 3 zeigt den Sortieralgorithmus *InsertionSort* in Form eines Struktogramms.

- Veranschaulichen Sie die Arbeitsweise des Algorithmus am Beispiel der Reihung
daten: 15, 13, 14, 18, 17, 11
- Begründen Sie, dass die Laufzeit der inneren Schleife des Algorithmus linear abhängig von der Länge n der Reihung `daten` ist, wenn diese unsortiert oder absteigend sortiert vorliegt. Untersuchen Sie auch, wie sich die Laufzeit der inneren Schleife ändert, wenn die Reihung `daten` bereits sortiert vorliegt.
- Bestimmen Sie die Komplexitätsklasse für die Laufzeit des Algorithmus im besten, im durchschnittlichen und im schlechtesten Fall.

insertionSort(daten: Reihung vom Inhaltstyp Ganzzahl):
Reihung vom Inhaltstyp Ganzzahl

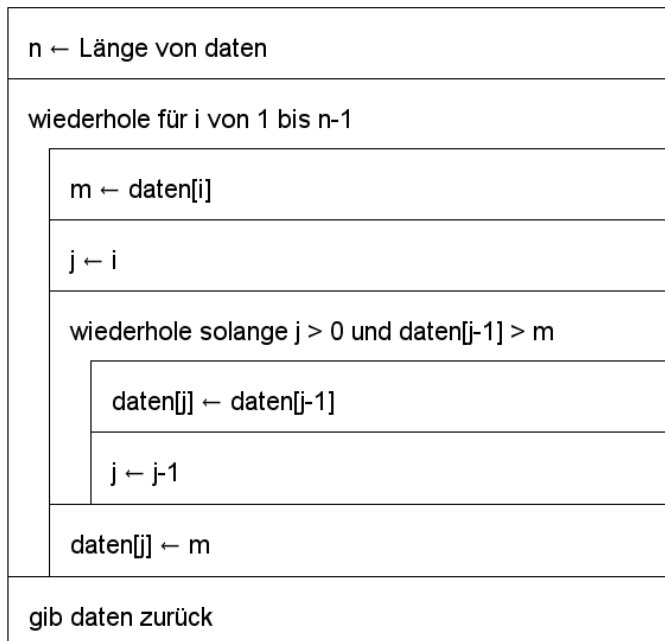


Abbildung 3: Struktogramm des Sortieralgorithmus *InsertionSort*

Aufgabe 4: Abbildung 4 und 5 zeigen den rekursiven Sortieralgorithmus *MergeSort* in Form eines Struktogramms. Der Algorithmus sortiert die Elemente in einer global definierten Reihung *daten* vom Inhaltstyp *Ganzzahl*. Abbildung 6 stellt das Sortierprinzip schematisch dar.

- Veranschaulichen Sie die Arbeitsweise des Algorithmus am Beispiel der Reihung *daten*: 15, 13, 14, 18, 17, 11 und dem initialen Aufruf `megeSort(0, 5)`.
- Begründen Sie, dass die Rekursionstiefe logarithmisch von der Länge *n* der Reihung *daten* abhängt.
- Begründen Sie, dass die Laufzeit für das Zusammenfügen der Elemente auf jeder Rekursionsebene mit der Hilfsoperation `merge` linear abhängig von der Gesamtanzahl *n* der zu sortierenden Elemente ist.
- Bestimmen Sie auf Basis Ihrer Überlegungen zu b) und c) die Komplexitätsklasse für die Laufzeit des Algorithmus *MergeSort*.

mergeSort(links: Ganzzahl, rechts: Ganzzahl)

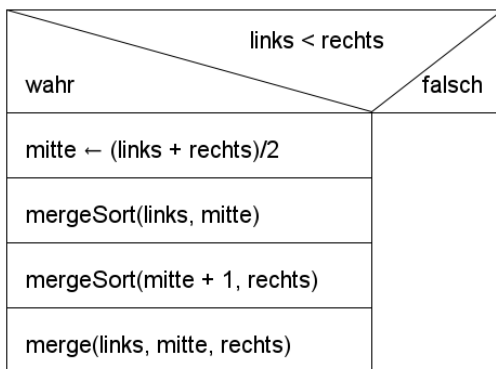


Abbildung 4: Struktogramm des rekursiven Sortieralgorithmus *MergeSort*

merge(links: Ganzzahl, mitte: Ganzzahl, rechts: Ganzzahl)

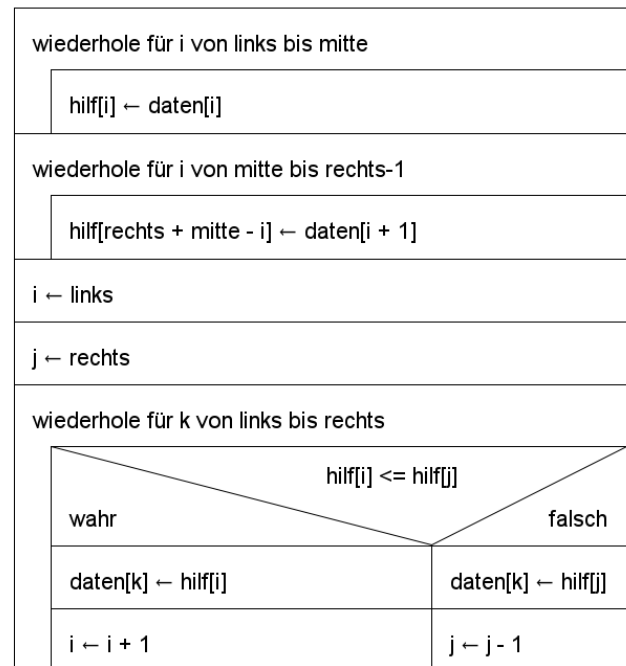


Abbildung 5: Hilfsoperation *merge* für den Sortieralgorithmus *MergeSort*

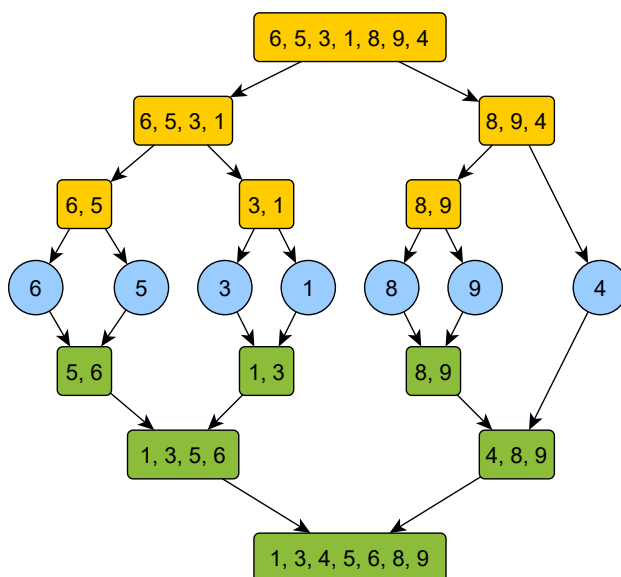


Abbildung 6: schematische Darstellung von *mergeSort*

Aufgabe 5:

- Stellen Sie Ihre Ergebnisse aus den Aufgaben 1 bis 4 in der Tabelle in Abbildung 7 zusammen.
- Bei der Auswahl eines Sortieralgorithmus ist manchmal auch von Interesse, ob ein Sortieralgorithmus stabil ist. Ein Sortieralgorithmus heißt stabil, wenn die Reihenfolge gleicher Werte beim Sortieren erhalten bleibt. Füllen Sie die Spalte *Stabilität* in Abbildung 7 entsprechend aus.
- Ein weiteres Kriterium bei der Auswahl eines Sortieralgorithmus ist der benötigte Speicherplatz. Untersuchen Sie, ob es hier Unterschiede zwischen den Sortieralgorithmen gibt.

Algorithmus	bester Fall	durchschnittlicher Fall	schlechtester Fall	Stabilität
SelectionSort				
BubbleSort				
InsertionSort				
MergeSort				

Abbildung 7: Zusammenfassung der Ergebnisse

Aufgabe 6: Entscheiden Sie für die folgenden Fälle jeweils begründet, welchen Sortieralgorithmus Sie verwenden würden.

- Ein Sportler erfasst jeden Tag, wie weit er bei 20 Trainingssprüngen gesprungen ist. Am Ende der Trainingseinheit sollen die Sprungweiten sortiert vom besten bis zum schlechtesten Sprung angezeigt werden.
- Ein Online-Shop hat bereits einen großen Kundenstamm und generiert zurzeit täglich 2 bis 3 neue Kunden. Die Kunden werden zunächst hinten an die Liste aller Kunden angehängt. Am Ende eines Tages wird die Kundenliste aktualisiert, indem sie alphabetisch nach den Namen der Kunden sortiert wird.
- Eine Supermarktkette mit ca. 3 Millionen registrierten Kunden bietet zwei Wochen lang ein Gewinnspiel an. Dabei wird täglich ein Foto mit einem Glas voll Reiskörner, Kaffeebohnen etc. veröffentlicht. Die Kunden können eine Schätzung abgeben, wie viele Reiskörner, Kaffeebohnen etc. sich in dem Behälter befinden. Am Ende eines jeden Spieltages sollen die Schätzungen der Kunden der Größe nach sortiert werden.

Aufgabe 7: In den Aufgaben 1 bis 5 wurde die Laufzeit der verschiedenen Sortieralgorithmen klassifiziert und aufgrund dieser Einordnung verglichen. In dieser Aufgabe soll die Laufzeit der verschiedenen Sortieralgorithmen einmal konkret gemessen werden. Beachten Sie dabei, dass der gemessene Wert nur exemplarisch ist und von verschiedenen Faktoren wie z. B. dem verwendeten Rechner, der konkreten Implementierung usw. abhängt.

- Analysieren Sie die Processing-Vorlage *Aufgabe7_Sortierverfahren*. Ergänzen Sie eine Implementierung der Sortierverfahren *SelectionSort*, *BubbleSort*, *InsertionSort* und *MergeSort*. Teilen Sie die Implementierung der Sortierverfahren unter sich auf.
- Verwenden Sie Ihr Programm aus a), um die Tabellen in Abbildung 8 mit Messwerten zu füllen. Sie können dabei arbeitsteilig vorgehen. Diskutieren Sie zunächst, welche Arbeitsaufteilung in der Gruppe sinnvoll ist und welche Absprachen vorher ggf. getroffen werden sollten.

- c) Interpretieren und erläutern Sie Ihre Messdaten. Stellen Sie Ihre Messdaten dazu ggf. auch grafisch dar, z. B. mithilfe eines Tabellenkalkulations-Programms.

unsortierte Reihung:

n Algorithmus	100	1000	10.000	100.000	200.000	500.000	1.000.000
SelectionSort							
BubbleSort							
InsertionSort							
MergeSort							

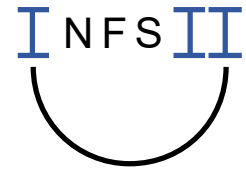
aufsteigend sortierte Reihung:

n Algorithmus	100	1000	10.000	100.000	200.000	500.000	1.000.000
SelectionSort							
BubbleSort							
InsertionSort							
MergeSort							

absteigend sortierte Reihung:

n Algorithmus	100	1000	10.000	100.000	200.000	500.000	1.000.000
SelectionSort							
BubbleSort							
InsertionSort							
MergeSort							

Abbildung 8: Laufzeitmessung für verschiedene Sortieralgorithmen



Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](#). Von der Lizenz ausgenommen ist das InfSII-Logo.

Für die korrekte Ausführbarkeit der beiliegenden Quelltexte wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.